Universidad Nacional
ARTURO JAURETCHE

TRABAJO FINAL

COMPLEJIDAD TEMPORAL

Carranza Braian

Comisión: 03

Introducción

Se nos pide a los estudiantes de Complejidad Temporal realizar diversas consultas sobre el juego *Who is who* (WiW), donde se tiene que adivinar primero cuál es el personaje elegido por el oponente. Nosotros tenemos que seleccionar uno de los personajes que nos de el programa, donde luego se empezara a jugar. En este caso se jugaría contra un Bot, donde cada uno tiene una lista de preguntas para realizarle al otro, todo esto por mediante turnos. El jugador que le toque responder, deberá responder con un "si" o "no", donde se ira descartando mediante la pregunta que se haga para encontrar al personaje seleccionado.

Objetivos

Implementar la estrategia que utiliza el Bot mediante el uso de un árbol de decisión. Estas mismas consultas, las haremos en la clase llamada "Estrategia".

Clases

Clasificador: es la que se encarga de procesar un conjunto de datos y determinar la mejor pregunta para dividirlos en dos grupos. Permite saber si en ese punto se debe crear una hoja o continuar creando nodos de decisión.

DecisionData: encapsula la información que se guarda en cada nodo del árbol binario de decisión. Puede almacenar dos tipos de datos:

- Una pregunta
- Una predicción

ArbolBinario<T>: representa la estructura de árbol binario genérico, donde cada nodo almacena un dato de tipo T y puede tener un hijo izquierdo y un hijo derecho.

Cola: es una estructura de datos genérica que permite almacenar y procesar elementos en orden de llegada. Tiene los métodos para encolar, desencolar, ver el primer elementos, si esta vacía la cola, contar sus elementos, etc.

Desarrollo

Sabemos que trabajamos con un árbol de decisión, ¿Qué quiere decir esto? Que a la hora de responder a las preguntas (verdadero o falso) el

árbol tomara una rama o la otra. Lo que nos pide en los métodos (consultas) son lo siguiente:

1. **CrearArbol (Clasificador clasificador):** este método construye un árbol de decisión a partir de un clasificador que es enviado como parámetro y retorna una instancia de **ArbolBinario<DecisionData>**

```
public ArbolBinario<DecisionData> CrearArbol(Clasificador clasificador)
{
    ArbolBinario<DecisionData> arbolNuevo;
    if(clasificador.crearHoja()){
        arbolNuevo=new ArbolBinario<DecisionData>(new DecisionData(clasificador.obtenerDatoHoja()));
    }
    else{
        arbolNuevo=new ArbolBinario<DecisionData>(new DecisionData(clasificador.obtenerPregunta()));
        arbolNuevo.agregarHijoIzquierdo(CrearArbol(clasificador.obtenerClasificadorIzquierdo())); //rama izquierda
        arbolNuevo.agregarHijoDerecho(CrearArbol(clasificador.obtenerClasificadorDerecho())); //rama derecha
    }

    return arbolNuevo;
}
```

¿Cómo funciona?

Si ya no tiene sentido seguir preguntando, ya que todos los datos dicen lo mismo, se crea una hoja con la respuesta final (por ejemplo: “es pepe”). Si todavía se puede dividir, se crea un nodo de pregunta, y dependiendo de la respuesta (si o no), se crea una rama izquierda (con datos que respondieron sí) y una rama derecha (con los datos que respondieron que no). Todo esto para que el “bot” tenga un árbol armado con preguntas y respuesta que le permitan ir adivinando personaje en el transcurso del juego.

El método inicia con la creación de un árbol completamente nuevo, donde al final del programa se retornará. Luego utilizamos el condicional “if” que si el clasificador no se puede dividir más (ósea, cuando la ganancia = 0), entonces se crea una hoja de árbol, que contiene los resultados de clasificaciones (*arbolNuevo=**new** ArbolBinario<DecisionData>(**new DecisionData**(clasificador.**obtenerDatoHoja**()));*).

Pero si todavía se puede dividir (hay ganancia), entonces se crea un nodo con la pregunta que representa la mejor división de los datos (*clasificador.obtenerPregunta()*), luego se crean los subárboles

izquierdo y derecho, de forma recursiva usando los clasificadores tanto el derecho como el izquierdo.

Por ejemplo: el primer clasificador encuentra la mejor pregunta que es “¿Tiene pelo rubio?”. Donde si la respuesta es “Sí”, crea un nuevo clasificador solo con los personajes que tienen pelo rubio y vuelve a preguntar (rama izquierda). Pero si la respuesta es “No”, hace lo mismo con los que no tienen pelo negro (rama derecha). Esto sigue hasta que en algún grupo ya no haya más división. Entonces se crea una hoja con los personajes que quedan y sus probabilidades.

2. **Consulta1 (ArbolBinario <DecisionData> arbol):** Retorna un texto con todas las posibles predicciones que puede calcular el árbol de decisión del sistema.

```
public String Consulta1(ArbolBinario<DecisionData> arbol)
{
    string resutl;
    if(arbol.esHoja()){
        resutl=arbol.getDatoRaiz().ToString();
    }

    else{
        resutl=Consulta1(arbol.getHijoIzquierdo());
        resutl=resutl+Consulta1(arbol.getHijoDerecho());
    }

    return resutl;
}
```

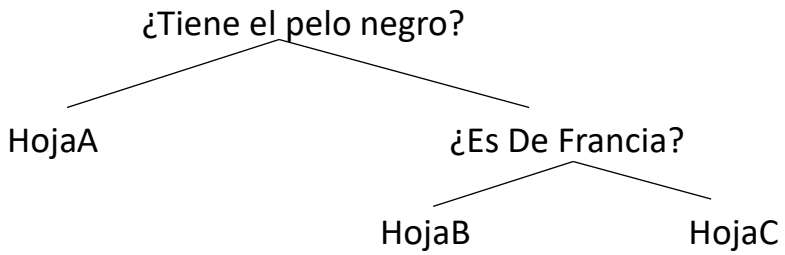
¿Cómo funciona?:
Si está en una hoja (respuesta final), guarda ese valor, si está en un nodo de pregunta, baja por la rama izquierda y por la derecha, y guarda las respuestas que encuentre. Va concatenando todo ese texto para mostrar al final todas las predicciones posibles. Es como una lista de todas las respuestas posibles que puede calcular el bot.

Este método recorre todo el árbol de decisión en forma recursiva y devuelve un string con los datos de las hojas (ósea, los resultados finales de las clasificaciones, como quien podría ser el personaje). Tenemos el caso Base (if) donde si el nodo actual es una hoja (no tiene hijos izquierdo ni derecho), entonces guarda en la variable “result” el dato que contiene esa hoja, convirtiéndola a String.

Sino es hoja (else), entonces hace una llamada recursiva para recorrer el hijo izquierdo, luego hace otra llamada recursiva para recorrer el hijo derecho, y al final concatenando los resultados de ambas llamadas en la variable result.
Al final de todo retorna el String armado con los datos de todas las hojas encontradas durante el recorrido.

Es decir, primero evalúa si es hoja, donde si es “si”, devuelve el dato, sino recorre recursivamente el hijo izquierdo y derecho, concatenando los resultados.

Por ejemplo:



- ¿Es hoja? En raíz: no
  - Va a un hijo izquierdo: HojaA – agrega HojaA
  - Va a hijo derecho ¿Es de Francia? – no es hoja
    - Va a hijo izquierdo: HojaB – agrega HojaB
    - Va a hijo derecho: HojaC – agrega HojaC

3. **Consulta2(ArbolBinario <DecisionData> arbol):** retorna un texto que contiene todos los caminos hasta cada predicción.

```
public String Consulta2(ArbolBinario<DecisionData> arbol)
{
    if(arbol.esHoja()){
        return "Prediccion: " + arbol.getDatoRaiz() + "\n";
    }
    else{
        string caminoIzquierdo=arbol.getDatoRaiz().ToString() + " -> Si ->" + Consulta2(arbol.getHijoIzquierdo());
        string caminoDerecho=arbol.getDatoRaiz().ToString() + " -> no ->" + Consulta2(arbol.getHijoDerecho());
        return caminoIzquierdo + caminoDerecho;
    }
}
```

Caso base: si el nodo actual es una hoja (no tiene hijos), devuelve el dato contenido en la hoja.
Si no es hoja, obtiene la pregunta actual (dato del nodo raíz), donde construye el camino hacia la izquierda:

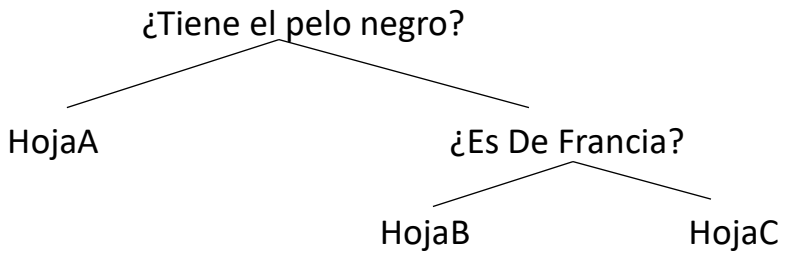
- Agrega la pregunta actual, agrega " -> Si ->" para indicar que está tomando la rama izquierda.
- Hace la llamada recursiva para seguir recorriendo desde el hijo izquierdo.

Construye el camino hacia la derecha:

- Agrega la pregunta actual, agrega " -> No ->", para tener de referencia que estamos en la rama derecha.
- Hace una llamad recursiva desde el hijo derecho.

Al final de todo concatena los dos caminos y los devuelve

Por ejemplo:



¿Tiene el pelo negro? –> Si -> Predicción: HojaA

¿Tiene el pelo negro? -> No -> ¿Es De Francia? -> Si -> Predicción_ HojaB

- ¿Es de Francia? -> No -> Predicción: Hoja C

4. **Consulta3 (ArbolBinario <DecisionData> arbol):** retorna un texto que contiene los datos almacenados en los nodos del árbol diferenciados por el nivel en que se encuentran.

```
public String Consulta3(ArbolBinario<DecisionData> arbol)
{
    Cola<ArbolBinario<DecisionData>> cola=new Cola<ArbolBinario<DecisionData>>();
    int nivel=0;
    cola.encolar(arbol);

    string result = "";

    while(!cola.esVacia()){
        int tamañoNivel=cola.cantidadElementos();
        result=result + "Nivel " + nivel + ":\n";

        for(int i=0; i<tamañoNivel; i++){
            ArbolBinario<DecisionData> nodoActual=cola.desencolar();
            result=result + nodoActual.getDatoRaiz() + "\n";

            if(nodoActual.getHijoIzquierdo() !=null){
                cola.encolar(nodoActual.getHijoIzquierdo());
            }
            if(nodoActual.getHijoDerecho() !=null){
                cola.encolar(nodoActual.getHijoDerecho());
            }
        }
```

```
            nivel++;
        }
        return result;
    }
```

Este método realiza un recorrido por niveles del árbol binario de decisiones, donde empieza con las inicializaciones, creando una instancia de la clase “Cola” para recorrer el árbol nivel por nivel, inicializando la variable nivel en 0, encolando la raíz del árbol, etc. Luego pasamos al bucle “While”: donde mientras la cola no esté vacía:

- Guarda cuántos nodos hay en este nivel (*tamañoNivel =cola.cantidadElementos()*)
- Agrega a la variable result el texto “Nivel X: \n ”
- Recorre todos los nodos de este nivel (bucle for hasta *tamañoNivel*)
    - Desencola el primer nodo
    - Agrega su dato (*GetDatoRaiz()*) a la variable *result.*
    - Si tiene hijo izquierdo, lo encola
    - Si tiene hijo derecho, lo encola
- Al terminar de procesar este nivel, incrementa la variable *nivel*

Es decir, que agrupa los datos de los nodos según el nivel que se encuentran. Donde se utiliza una cola que encola los nodos de cada nivel y luego, uno a uno los procesa, agregando sus datos a una cadena de texto con su nivel correspondiente. Si un nodo tiene hijos, se encolan para ser procesados en el siguiente nivel, esto se repite hasta recorrer todos los nodos del árbol.

Posibles mejoras

Unas de las mejoras que se me ocurren son la de un método que devuelva la altura del árbol, para saber cuántas preguntas máximo podría hacer el Bot antes de adivinar

Conclusiones

La conclusión que llego de este proyecto, es lo que podemos hacer con los recorridos, en la primera clase de la materia vimos lo que fueron los recorridos de preorden, de orden y de nivel, donde con eso ya podíamos realizar el proyecto, no basto más que eso, es decir, lo que se puede realizar con los caminos es muy amplio y mucha utilidad.